-EXHIBIT 2-
To Defendant Ameriprise's
Renewed Motion to Dismiss
XimpleWare v. Versata, et al [13-cv-5160-SI]

VTD-XML: The Future of XML

Frequently Asked Questions

*What is VTD-XML designed for ?

An important design goal of VTD-XML is to significantly improve the processing performance of XML so that network switches and routers can inspect XML content at very high speed. Also, VTD-XML is designed as a general-purpose replacement of DOM, SAX or Pull parsing.

*How does VTD-XML solve XML's performance issue?

Well, XML's often mentioned issues are size and performance. VTD-XML doesn't solve XML's verbosity issue

well. XML's often mentioned issues are size and performance. VTD-XML doesn't solve XML's verbosity issue. In terms of performance, XML never had a performance issue, it is XML's processing models that have performance issues. In more details, the following points can be made:

- Performance and memory usage issues are specific to DOM, not XML. Using DOM to process any data, such as SGML or binary XML, will result in the same issues.
- The usability issue is specific to SAX, not XML. Using SAX to process any data, such as HTML or binary XML, will result in the same issue.
- VTD-XML solves the performance issue at the fundamental level-- to improve XML processing. Using VTD, one can improve virtually any data processing, such as HTML, or Binary XML.

In the hands of skilled developers, what VTD-XML can achieve is nothing short of stunning. We have routinely heard success stories of someone cutting down the processing time by 10x~100x(we are not joking).

* Is VTD-XML conformant?

VTD-XML fully conforms to W3C XML 1.0 recommendation with the exception of DTD and external entity. It passes all the relevant conformance tests.

VTD-XML (since v 2.9) fully conforms to XML Namespace 1.0 spec.

VTD-XML fully conforms with W3C XPath 1.0 recommendation. It supports all the built-in functions. It also supports many XPath 2.0 functions.

* Who are using VTD-XML?

Telecom companies, insurance companies, financial institutions, governments, and companies and organizations of various types and purposes world-wide, with more than 60,000 downloads, it is difficult to enumerate what and how people are using VTD-XML.

* What kind of build environment has VTD-XML been tested under?

Java (1.2, 1.3, 4, 5, 6), Open JDK, .NET 1.1,2.0,3.0..., Mono and (mingw, VC++ , GCC, ...).

*How do I get started learning VTD-XML?

There are a few resources that help you get started.

- VTD-XML introduction and API overview
- The code samples section of VTD-XML project page
- VTD-XML user's guide
- Various articles and presentations
- The VTD-XML blog

*How does VTD-XML differ from DOM and SAX?

Comparing with DOM, VTD-XML is significantly faster (up to 10x), more memory-efficient (up to 5x).

Comparing with SAX/PULL, VTD-XML is not only faster, but also is capable of random-access, therefore is easier to use.

* What are the maximum file size supported by VTD-XML?

For VTD-XML's regular version, it depends on the name space enablement.

- When namespace is not enabled, the maximum file size is 2 GB.
- When namespace is enabled, the maximum file size is 1 GB.

With VTD-XML's extended edition, the supported maximum file size is 256 GB, regardless of namespace support.

* Why is VTD-XML uniquely suited for implementing SOA applications?

Simply put, while each of its benefits stands out among the crowd of XML technologies, it is the combination of VTD-XML's attributes and benefits that make VTD-XML the ultimate XML technology for building next generation, data-centric, loosely-couple SOA and Cloud applications. And no other XML

technologies come close.

At a deeper level, VTD-XML possibly marks the start of a paradigm shift in which (1) "modeling everything as an object" is no longer a knee-jerk reaction -- it is done only when it makes sense; and (2) OO-based approaches to building distributed applications should finally given way to data-centric way of building distributed applications.

* Can you explain the GPL license a bit more?

The GPL does not necessarily require one to disclose their source code when modifying a GPL-covered work or using GPL-covered code in a new work. This requirement arises only when the new project is "distributed" to a third parties. If the resulting software is kept only for us by the modifier, no disclosure of source code is required.

If you don't like the restriction of GPL, XimpleWare also offers flexible commercial licenses for VTD-XML. Please email us at sales@ximpleware.com for more details.

* Why doesn't VTD-XML support DTD and external entities?

There are two ways to look at this XML spec 1.0. One way is that it is an article created by visionaries to tell the world how things should be done. XimpleWare views it the other way: XML 1.0 actually evolves and we believe that some parts of the spec, such as DTD and external entities, have in effect been deprecated because they are complex but not very useful.

*Is VTD-XML an indexer or a parser?

VTD-XML is simultaneously a parser and an indexer in that VTD-XML's API make direct use of VTD index to achieve the purpose of parsing.

*What are the optimizations that VTD-XML has implemented?

Every time an object is created, it needs to be garbage-collected. So there is a round trip penalty.

Every time one takes apart the the document for a small change, he will have to put everything back together. So there is another round trip penalty.

Every time one decodes (e.g. from UTF-8 to UCS 2) the entire document for a small change, he will have to encode the document when writing out on disk. So there is yet another roundtrip penalty.

Putting all these overheads together, XML processing performance probably isn't going to be very good.

VTD-XML is designed from ground up to overcome these overheads.

The first thing VTD-XML does is to keep the document intact in memory, and un-decoded. The tokenization is done by only recording the starting offset and length.

Next, VTD-XML represents tokens in 64 bit integers (VTD records). Because VTD records are constant in length, they can be stored in large memory blocks, resulting in a very significant memory saving.

Finally, VTD-XML's internal representation is inherently persistent, making possible "parse once; use many-times".

*How much memory does VTD-XML use?

Typically around 1.3~1.5x of the size of the document, with 1 being the document itself. In the extended version, developers can choose to memory-map the XML document, making it possible to process XML files whose size is bigger than available physical memory.

* Why should I use VTD-XML for large XML files?

For numerous reasons summarized below:

- Performance: The performance of VTD-XML is far better than SAX
- Ease to use: Random access combined with XPath makes application easy to write
- Better maintainability: App code is shorter and simpler to understand.
- Incremental update: Occasional, small changes become very efficient.
- Indexing: Pre-parsed form of XML will further boost processing performance.
- Memory Mapping: Extended VTD-XML has the option to memory-map XML documents, making it possible to process XML files whose size is bigger than available physical memory.

- possible to process XML files whose size is bounded only by available memory.
- Other features: Cut, paste, split and assemble XML documents is only possible with VTD-XML.

In order to take advantage of VTD-XML, we recommended that developers split their ultra large XML documents into smaller, more manageable chucks (<2GB).

Since VTD-XML 2.4, XimpleWare introduces an extended version of VTD-XML capable of processing XML documents up to 256 GB in size.

*How does VTD-XML deal with various encodings?

Our approach is to build intelligence into various record-to-string comparison functions so that one deals with a token (the index of the VTD record) without knowing the underlying document encoding format.

*What encoding types does VTD-XML support?

As of Version 2.6, VTD-XML supports ASCII, UTF-8, UTF-8859-1 thru UTF-8859-16, Win1250 thru WIN1258, UTF-16LE, and UTF-16BE.

*How does VTD-XML achieve random access?

By using Location Caches (LC), which are essentially hierarchical element directories.

LCs are allocated on a per-level basis; i.e., the same LC indexes all elements of the same nesting depth. A LC entry is a 64-bit integer whose upper 32 bits is the index value of a VTD record of an element (starting tag), and whose lower 32 bits point to the LC entry corresponding to the first child element.

The motivation is to stick with the winning strategy of VTD: constant token length and inherent persistence.

The result: LC cost is about *10%* of the total VTD-XML processing cost.

*Why element-based hierarchy?

After VTD records are generated, related records, such as an element and its attributes, are associated with each other by their adjacency. Element-based indexing just seems the natural thing to do.

*How does VTD-XML compare to DOM?

VTD-XML is not DOM in that VTD-XML has a different API not based on DOM nodes. VTD-XML uses a cursor to move to different locations in the XML hierarchy. The core methods are "toElement()" and "toElementNS()" reflective of VTD-XML's element-only hierarchy.

The write feature of VTD-XML emphasizes the modification of XML; DOM, on the other hand, modifies the data structure.

Also VTD-XML breaks new grounds in that it allows direct byte level manipulation of XML documents. Operation such as cutting, pasting and splitting becomes very efficient with VTD-XML.

* If I get a VTD index value of an attribute name, how do I retrieve the attribute value?

If the attribute name's index value is i, then attribute value corresponds to i+1.

*Why 64 bits?

For a 32-bit OS, 64 bits provide enough room to describe a VTD record.

Also consider the following:

- Data bus of a PC is 64-bit wide.
- PCI-64, PCI-X, and PCI-Express all use 64-bit data buses.
- Many segment registers of IA-32 are 64-bit.
- IA-64 and EM64 are all 64-bit architecture.

* What is Buffer Reuse?

It's a new feature introduced in version 1.5 of VTD-XML. Basically, because VTD-XML internally use large

memory buffers as VTD record containers, they are totally reusable: those memory buffers can be allocated once and use for as long as the applications like. It is like a bucket, you can use it to fill water, pour it out, then refill. To use this feature, just use the setDoc_BR() (instead of setDoc()) of VTDGen.

*Why is VTD-XML well suited for hardware implementation?

Two reasons:

1. VTD and LC are inherent persistent
2. The persistence is post binding.

*Why not DOM on a chip?

DOM on a chip doesn't work well.

The main reason is that the DOM makes heavy use of pointers, making it unsuitable for inter-process data transfer- something that must be done after the chip finishes processing XML.

*Why not SAX on a chip?

SAX on a chip doesn't make sense because SAX parsing per se is usually not the performance bottleneck.

*How does the C version of VTD-XML compare with the Java Version?

Well, C version is actually a hair (<5%) slower than VTD-XML's Java version running on server JVM, which does a phenomenal job optimizing the byte code. For that reason, we highly recommend that the server JVM should be used for all purposes.

In terms of the implementation, C is almost an replica of the Java version in that virtually for every line of C code, there is a corresponding line of Java code that performs the same function. The VTD-XML project team chose to do so to minimize the porting effort because C and Java syntax have a large number of overlaps. Another benefit is that if there is a bug in Java, we can be certain there is going to be an exact bug in the same place in C code.

*Is there a plan to offer a SAX/StAX interface on top of VTD?

This is a very interesting and recurring topic that we routinely discuss with open source developers and XML enthusiasts. And there are actually no technical limitations to create a SAX/StAX compatible interface. But consider the following:

- VTD-XML's performance and usability is far superior to forward-only SAX/StAX.
- Implementing SAX on top of VTD loses SAX's only advantage: processing XML bigger than the available physically memory.

So there is little incentive to offer SAX or StAX over VTD.

*Is there a plan to offer DOM interface over VTD?

Well this is another interesting topic. DOM, as a W3C spec, predates XML. DOM's memory and processing overheads are caused by its excessive use of objects. Furthermore, DOM's API design is based on the assumptions that objects are the only way to represent nodes. So creating a DOM-compatible interface requires VTD-XML allocate a lot of objects just like DOM, making it difficult, if not impossible, to optimize performance and memory usage and essentially defeating the purpose of VTD in the first place.

*Although VTD-XML claims to be non-extractive, but in many cases developers still have to extract data, will that degrade performance of VTD-XML?

No, that is actually no much of a problem. For the following reasons:

1. A lot of metadata, i.e. tags and attribute names, are mostly used for navigation purposes, so they don't need to be converted into string objects
2. VTD-XML converts VTD records to primitive data types without converting them into strings.
3. In DOM the biggest overhead is creating node objects, which VTD-XML completely avoids.
4. Even one has to extract data into strings, if he knows beforehand the length of the string, the string allocation is in fact quite fast. If string length is not known, the string buffer implementation

potentially makes a lot of copy and discard if the string length exceeds allocated buffer length, which can be inefficient. A VTD record encodes the token length, which improve string allocation performance.

*When compiling the C version of VTD-XML, I see lots of warning messages. Is there anything wrong?

To emulate Java's interface, the C version of VTD-XML use "structs" and functional pointers that cause those warning message. So it is ok, you can safely ignore those warning messages.

*What do I have to do differently to get VTD-XML compiled using Visual Studio?

You need to set the compiler to C mode. Following the menu selection shown below:

Configuration properties> C/C++> *advanced* > *compile as* > *Compile as C code*